Automatic generation of sources lemmas in Tamarin

Jannik Dreier

joint work with Véronique Cortier and Stéphanie Delaune

GDR Winter School
The Internet – February 10, 2021
Interaction and Automation

Tamarin’s **interactive mode** allows the user to inspect and direct proof search

- Gives the **flexibility** required for complex case-studies
- Enables **fine-tuning** of models and proof strategies

On the downside, Tamarin’s **automatic mode** often fails (compared to, e.g., ProVerif), even on relatively **simple examples**.
Tamarin’s **interactive mode** allows the user to inspect and direct proof search

- Gives the **flexibility** required for complex case-studies
- Enables **fine-tuning** of models and proof strategies

On the downside, Tamarin’s **automatic mode** often fails (compared to, e.g., ProVerif), even on relatively **simple examples**.

One of the **main reasons**: **partial deconstructions**.

Our **contribution**: **automatic handling of partial deconstructions** in most cases.
1. Introduction
2. Partial deconstructions
3. Algorithm
4. Implementation and evaluation
5. Conclusion
1. Introduction

2. Partial deconstructions

3. Algorithm

4. Implementation and evaluation

5. Conclusion
Consider the following toy protocol between the initiator 🧑 and the responder 🧑:

1. 🧑 → 🧑: \( \{ \text{req, } l, n \}_{\text{pk}(R)} \)
2. 🧑 → 🧑: \( \{ \text{rep, } n \}_{\text{pk}(l)} \)
Consider the following toy protocol between the **initiator** and the **responder**:

1. \( \text{ initiator } \rightarrow \text{ responder } : \{ \text{req, I, n} \}_{pk(R)} \)
2. \( \text{ responder } \rightarrow \text{ initiator } : \{ \text{rep, n} \}_{pk(I)} \)

In **Tamarin** the **initiator** can be modeled using the following **rule**:

```latex
\text{rule Rule_I:}
\begin{align*}
[ \ Fr(n), \\
!Pk(R, pkR), \\
!Ltk(I, ltkI) ] \\
\text{--}[ \text{ SecretI(I, R, n) }]\rightarrow \\
[ \text{ Out(aenc{'req', I, n}pkR) } ]
\end{align*}
```
Consider the following toy protocol between the initiator  🟦 and the responder 🟤:

1. 🟦 → 🟤: \{req, I, n\}_{pk(R)}
2. 🟤 → 🟦: \{rep, n\}_{pk(I)}

The responder can be modeled using the following **rule**:

```
rule Rule_R:
[ In(aenc{’req’, I, x}pk(ltkR)),
  !Ltk(R, ltkR),
  !Pk(I, pkI) ]
-->[ ]->
[ Out(aenc{’rep’, x}pkI) ]
```
Consider the following toy protocol between the **initiator** 🍃 and the **responder** 🌿:

1. 🍃 → 🌿: \{req, I, n\}_{pk(R)}
2. 🌿 → 🍃: \{rep, n\}_{pk(I)}

**Secrecy** for the nonce \( n \) can be modeled using the following **lemma**:

```latex
lemma nonce_secrecy:
"not(
    \exists A B s \#i. SecretI(A, B, s) @ i \\
    \& (\exists \#j. K(s) @ j)
)
"
```
Consider the following toy protocol between the initiator 🌸 and the responder 🌻:

1. 🌸 → 🌻: \{req, l, n\}_{pk(R)}
2. 🌻 → 🌸: \{rep, n\}_{pk(l)}

Secrecy for the nonce $n$ can be modeled using the following lemma:

lemma nonce_secrecy:

"not(
    Ex A B s #i. SecretI(A, B, s) @ i
    & (Ex #j. K(s) @ j)
"

Unfortunately, the proof of this lemma does not terminate due to partial deconstructions.
Partial deconstructions

**TAMARIN** pre-computes all possible origins (called **sources**) of all protocol and intruder facts.

This can stop in an incomplete stage (called **partial deconstruction**) if **TAMARIN** lacks sufficient information about the origins of some fact(s).
**Partial deconstructions**

**Tamarin** pre-computes all possible origins (called **sources**) of all protocol and intruder facts.

This can stop in an incomplete stage (called **partial deconstruction**) if **Tamarin** lacks sufficient information about the origins of some fact(s).

To **resolve** these partial deconstructions, one has to write a **sources lemma** detailing the possible origins of the problematic fact(s).

Sources lemmas are used to **refine** the sources, but they also need to be **proven correct**.
Example: Partial deconstruction
Example: Partial deconstruction
We know that the input is either the message sent by the initiator, or a message constructed by the intruder.
We **know** that the **input** is either the message sent by the initiator, or a message **constructed** by the intruder.

Need to **annotate** the protocol rules:

```plaintext
rule Rule_I:
  [ Fr(n), !Pk(R, pkR), !Ltk(I, ltkI) ]
  --[ I(aenc{'req', I, n}pkR), SecretI(I, R, n) ]->
  [ Out(aenc{'req', I, n}pkR) ]

rule Rule_R:
  [ In(aenc{'req', I, x}pk(ltkR)),
    !Ltk(R, ltkR), !Pk(I, pkI) ]
  --[ R(aenc{'req', I, x}pk(ltkR), x) ]->
  [ Out(aenc{'rep', x}pkI) ]
```
We know that the input is either the message sent by the initiator, or a message constructed by the intruder.

Need to annotate the protocol rules:

rule Rule_I:

\[
[ \text{Fr}(n), \text{!Pk}(R, \text{pkR}), \text{!Ltk}(I, \text{ltkI})] \\
\longrightarrow [ \text{I}(\text{aenc}\{\text{req}, I, n\}\text{pkR}), \text{SecretI}(I, R, n) ] \rightarrow [ \text{Out}(\text{aenc}\{\text{req}, I, n\}\text{pkR}) ]
\]

rule Rule_R:

\[
[ \text{In}(\text{aenc}\{\text{req}, I, x\}\text{pk(ltkR)}), \text{!Ltk}(R, \text{ltkR}), \text{!Pk}(I, \text{pkI})] \\
\longrightarrow [ \text{R}(\text{aenc}\{\text{req}, I, x\}\text{pk(ltkR)}, x) ] \rightarrow [ \text{Out}(\text{aenc}\{\text{rep}, x\}\text{pkI}) ]
\]

Source lemma:

lemma typing [sources]:

"All x m #i. R(m,x)@#i ==> ((Ex #j. I(m)@#j & #j < #i) \\
| (Ex #j. KU(x)@#j & #j < #i))"
**Generalize idea & automate** the approach:

1. Inspect the **raw sources** computed by **TAMARIN**
2. For each partial deconstruction:
   1. Identify the **variables** and **facts** causing the partial deconstruction
   2. Identify rules producing **matching conclusions**
   3. Add necessary **annotations** to the concerned rules
3. Generate a **sources lemma** using all annotations and add it to the theory

Note that TAMARIN will verify the correctness of the generated lemma. But we actually proved that the lemmas we generate are correct under some assumptions (well-formed rules, subterm-convergent equational theory).
Generalize idea & automate the approach:

1. Inspect the raw sources computed by Tamarin
2. For each partial deconstruction:
   1. Identify the variables and facts causing the partial deconstruction
   2. Identify rules producing matching conclusions
   3. Add necessary annotations to the concerned rules
3. Generate a sources lemma using all annotations and add it to the theory

Note that Tamarin will verify the correctness of the generated lemma.

But we actually proved that the lemmas we generate are correct under some assumptions (well-formed rules, subterm-convergent equational theory).
How to identify matching conclusions?

First idea
Extract input message and try to unify with all outputs.
• Turns out to be insufficient, consider following example:
  • Input: \( \langle \text{enc}(a, k_1), \text{enc}(b, k_2) \rangle \)
  • Output 1: \( \text{enc}(a, k_1) \)
  • Output 2: \( \text{enc}(b, k_2) \)
  • Unification fails, but the intruder can easily compose both outputs

Solution
Use protected subterms:
• A protected subterm is subterm whose head symbol is neither a pair nor an AC symbol
• Allows us to abstract away pairs
Identifying matching conclusions

- Extract the **deepest** protected subterms **containing the variable** causing the partial deconstruction from the **facts** in the raw source.

**Example**

\[ t = \text{enc}(\langle x, \text{enc}(\langle b, x \rangle, k_2) \rangle, k_1) \]

has two deepest protected subterms w.r.t. \( x \):

\[
\text{enc}(\langle b, x \rangle, k_2) \quad \text{and} \quad \text{enc}(\langle x, \text{enc}(\langle b, x \rangle, k_2) \rangle, k_1)
\]

- Extract **all protected subterms** from all conclusions of all rules and try to **unify** with the deepest protected subterms.
- If unification succeeds, we have a **match**.
Identifying matching conclusions

• Extract the **deepest** protected subterms containing the **variable** causing the partial deconstruction from the **facts** in the raw source

**Example**

\[ t = \text{enc}(\langle x, \text{enc}(\langle b, x \rangle, k_2) \rangle, k_1) \]

has two deepest protected subterms w.r.t. \( x \):

\[ \text{enc}(\langle b, x \rangle, k_2) \quad \text{and} \quad \text{enc}(\langle x, \text{enc}(\langle b, x \rangle, k_2) \rangle, k_1) \]

• Extract all **protected subterms** from all conclusions of all rules and try to **unify** with the deepest protected subterms

• If unification succeeds, we have a **match**.
Identifying matching conclusions

- Extract the **deepest** protected subterms containing the **variable** causing the partial deconstruction from the **facts** in the raw source.

**Example**

\[ t = \text{enc}(\langle x, \text{enc}(\langle b, x \rangle, k_2) \rangle, k_1) \]

has two deepest protected subterms w.r.t. \( x \):

\[ \text{enc}(\langle b, x \rangle, k_2) \quad \text{and} \quad \text{enc}(\langle x, \text{enc}(\langle b, x \rangle, k_2) \rangle, k_1) \]

- Extract **all protected subterms** from all conclusions of all rules and try to **unify** with the deepest protected subterms.
- If unification succeeds, we have a **match**.
Plan

1. Introduction
2. Partial deconstructions
3. Algorithm
4. Implementation and evaluation
5. Conclusion
Implementation

We implemented the algorithm in Tamarin (available in version 1.6.0).

To enable automatic source lemma generation, run Tamarin with --auto-sources:

• If partial deconstructions are present and there is no sources lemma, the algorithm generates a lemma and adds it to the theory.

• If there is already a lemma, or there are no partial deconstructions, Tamarin runs as usual.

• If a protocol rule has multiple variants, our algorithms considers all variants individually.
We tried numerous examples from the **SPORE library**:

<table>
<thead>
<tr>
<th>Protocol Name</th>
<th>Partial Dec.</th>
<th>Resolved</th>
<th>Automatic</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew Secure RPC</td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>42.8s</td>
</tr>
<tr>
<td>Modified Andrew Secure RPC</td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>134.3s</td>
</tr>
<tr>
<td>BAN Concrete Andrew Secure RPC</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>10.6s</td>
</tr>
<tr>
<td>Lowe modified BAN Andrew Secure RPC</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>29.8s</td>
</tr>
<tr>
<td>CCITT 1</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>0.8s</td>
</tr>
<tr>
<td>CCITT 1c</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>1.2s</td>
</tr>
<tr>
<td>CCITT 3</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>186.1s</td>
</tr>
<tr>
<td>CCITT 3 BAN</td>
<td>0</td>
<td>-</td>
<td>✓</td>
<td>3.7s</td>
</tr>
<tr>
<td>Denning Sacco Secret Key</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.8s</td>
</tr>
<tr>
<td>Denning Sacco Secret Key - Lowe</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>2.7s</td>
</tr>
<tr>
<td>Needham Schroeder Secret Key</td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>3.6s</td>
</tr>
<tr>
<td>Amended Needham Schroeder Secret Key</td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>7.1s</td>
</tr>
<tr>
<td>Otway Rees</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>7.7s</td>
</tr>
<tr>
<td>SpliceAS</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>5.9s</td>
</tr>
<tr>
<td>SpliceAS 2</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>7.3s</td>
</tr>
<tr>
<td>SpliceAS 3</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>8.7s</td>
</tr>
<tr>
<td>Wide Mouthed Frog</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.6s</td>
</tr>
<tr>
<td>Wide Mouthed Frog Lowe</td>
<td>14</td>
<td>✓</td>
<td>✓</td>
<td>3.5s</td>
</tr>
<tr>
<td>WooLam Pi f</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.6s</td>
</tr>
<tr>
<td>Yahalom</td>
<td>15</td>
<td>✓</td>
<td>✓</td>
<td>3.1s</td>
</tr>
<tr>
<td>Yahalom - BAN</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>0.9s</td>
</tr>
<tr>
<td>Yahalom - Lowe</td>
<td>21</td>
<td>✓</td>
<td>✓</td>
<td>2.2s</td>
</tr>
</tbody>
</table>
Case studies: Tamarin repository

We also tested all examples from the Tamarin repository that contained partial deconstructions:

<table>
<thead>
<tr>
<th>Name</th>
<th>Partial Dec.</th>
<th>Resolved</th>
<th>Automatic</th>
<th>Time (new)</th>
<th>Time (previous)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feldhofer (Equivalence)</td>
<td>5</td>
<td>✓</td>
<td>✓</td>
<td>3.8s</td>
<td>3.5s</td>
</tr>
<tr>
<td>NSLPK3</td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>1.8s</td>
<td>1.8s</td>
</tr>
<tr>
<td>NSLPK3 untagged</td>
<td>12</td>
<td>✓</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NSPK3</td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>2.4s</td>
<td>2.2s</td>
</tr>
<tr>
<td>JCS12 Typing Example</td>
<td>7</td>
<td>✓</td>
<td>x</td>
<td>0.3s</td>
<td>0.2s</td>
</tr>
<tr>
<td>Minimal Typing Example</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td>0.1s</td>
<td>0.1s</td>
</tr>
<tr>
<td>Simple RFID Protocol</td>
<td>24</td>
<td>✓</td>
<td>x</td>
<td>0.7s</td>
<td>0.5s</td>
</tr>
<tr>
<td>StatVerif Security Device</td>
<td>12</td>
<td>✓</td>
<td>✓</td>
<td>0.3s</td>
<td>0.4s</td>
</tr>
<tr>
<td>Envelope Protocol</td>
<td>9</td>
<td>✓</td>
<td>x</td>
<td>25.7s</td>
<td>25.3s</td>
</tr>
<tr>
<td>TPM Exclusive Secrets</td>
<td>9</td>
<td>✓</td>
<td>x</td>
<td>1.8s</td>
<td>1.8s</td>
</tr>
<tr>
<td>NSL untagged (SAPIC)</td>
<td>18</td>
<td>✓</td>
<td>✓</td>
<td>4.3s</td>
<td>19.9s</td>
</tr>
<tr>
<td>StatVerif Left-Right (SAPIC)</td>
<td>18</td>
<td>✓</td>
<td>✓</td>
<td>28.8s</td>
<td>29.6s</td>
</tr>
<tr>
<td>TPM Envelope (Equivalence)</td>
<td>9</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5G AKA</td>
<td>240</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Alethea</td>
<td>30</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PKCS11-templates</td>
<td>68</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NSLPK3XOR</td>
<td>24</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Chaum Offline Anonymity</td>
<td>128</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FOO Eligibility</td>
<td>70</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Okamoto Eligibility</td>
<td>66</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
• For all examples from SPORE, our approach was successful in resolving the partial deconstructions, and the entire verification became automatic.

• In most examples from the Tamarin repository, our approach was also successful, including examples with equivalence properties or generated by SAPIC. Verification times were similar to manual source lemmas.

• In some cases the partial deconstructions were resolved but the rest was not automatic: further intermediate lemmas or other annotations were required.

• Our approach failed for three reasons:
  • A too complex equational theory (not subterm convergent, AC symbols, . . .)
  • Partial deconstructions caused by state facts rather than messages
  • Tamarin fails to prove the generated sources lemma
1. Introduction

2. Partial deconstructions

3. Algorithm

4. Implementation and evaluation

5. Conclusion
• Automation in Tamarin often fails because of partial deconstructions
• Developed & implemented a new algorithm to automatically generate sources lemmas
• Proved correctness of the generated lemmas
• Algorithm works well in practice, many examples become fully or at least partly automatic
• Available in Tamarin 1.6.0
• Future work:
  • Handle more general equational theories
  • Handle partial deconstructions stemming from state facts (work in progress)